Machine Learning W5. Neural Network Learning.

1. Cost Function.

- neural network for classification.
- L: total # of layers.
- s_l: # of units in layer l.

(1) Binary classification.

$y=0$ or 1. ; 1 output unit. $h_\Theta(x) \in \mathbb{R}$.

$s_L=1$, $K=1$.

(2) Multi-class classification.

$y \in \mathbb{R}^K$. e.g. $\begin{pmatrix}1\\0\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\0\\0\end{pmatrix}, \dots$

K output units.

$h_\Theta(x) \in \mathbb{R}^K$. $s_L = K$. $(K \geq 3)$

Cost function

Logistic regression. $J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \cdot \log\left(1-h_\theta(x^{(i)})\right)\right]$

↓ generalized.

$+\frac{\lambda}{m}\sum_{j=1}^{n}\theta_j^2$ — regularization.

Neural network. $h_\Theta(x) \in \mathbb{R}^K$; $(h_\Theta(x))_i = i^{th}$ output.

$$J_\Theta(x) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \cdot \log\left(h_\Theta(x^{(i)})\right)_k + (1-y_k^{(i)}) \log\left(1-h_\Theta(x^{(i)})\right)_k\right]$$

sum over output units ←

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\theta_{ji}^{(l)}\right)^2$$

2. Back Propagation Algorithm

$\min_{\Theta} J(\Theta)$ → need to compute $J(\Theta)$

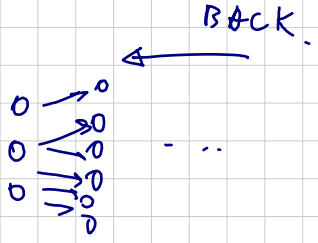
$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

(1) Gradient computation

Consider only a single training example (x, y)

Forward propagation.

- $a^{(1)} = x$
  $z^{(2)} = \Theta^{(1)} a^{(1)}$
- $a^{(2)} = g(z^{(2)})$ (add $a_0^{(2)}$)
- ⋮
- $a^{(4)} = h_\Theta(x) = g(z^{(4)})$



Back propagation

intuition. $\delta_j^{(l)}$ = "error" of node j in layer l.

recall: $a_j^{(l)}$ activation unit in ..

for each output unit. (layer $L=4$)

$$\delta_j^{(4)} = a_j^{(4)} - y_j \quad \text{← "labelled" training set.}$$

vectorized.

$$\underline{\delta}^{(4)} = \underline{a}^{(4)} - \underline{y} \qquad \in \mathbb{R}^K \quad (K: \#\text{ of output unit.})$$

back.

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)}) \qquad g'(z^{(3)}) = \underline{a}^{(3)} .* (1 - \underline{a}^{(3)})$$

$$\delta^{(2)} = (\Theta^{(2)})^T \cdot \delta^{(3)} .* g'(z^{(2)}) \qquad g'(z^{(2)}) = \underline{a}^{(2)} .* (1 - \underline{a}^{(2)})$$

(

$$g = \frac{1}{1 + e^{a^T x}}.$$

It can be shown that:

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \cdot \delta_i^{(l+1)}$$

(for $\lambda = 0$)

Backpropagation Algorithm

Training set $\{ (x^{(1)}, y^{(1)}) \cdots (x^{(m)}, y^{(m)}) \}$

Set $\Delta_{ij}^{(l)} = 0$ for all l, i, j (→ used to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$)

For $i = 1 : m$ %. loop thru. training set.

- ① set $a^{(1)} = x^{(i)}$ → input for i^{th} training example.
- ② perform forward propagation to compute $a^{(l)}$ for $l = 1, 2, 3 \cdots L$
- ③ Using $y^{(i)}$, $\delta^{(L)} = a^{(L)} - y^{(i)}$

  $\delta^{(L-1)}, \delta^{(L-2)} \cdots$

  $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \cdot \delta_i^{(l+1)}$

  ↓ vect.

  $\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

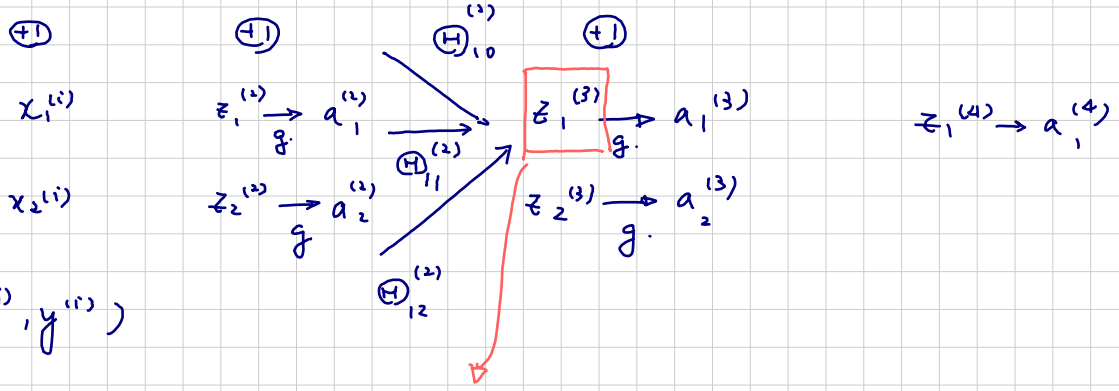
$$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \quad \text{if } j \neq 0$$

$$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} \quad \text{if } j = 0$$

$$\frac{\partial}{\partial \Theta_{ij}} J(\Theta) = D_{ij}^{(l)}$$

3. Backpropagation Intuition.

(1) Forward propagation.



Now, notice that backpropagation is doing something very similar to this, but in the reversed dir. ($\longleftarrow$) instead of ($\longrightarrow$)

(2) What is back propagation doing?

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \cdot \log\left[h_\Theta(x^{(i)})\right] + (1-y^{(i)}) \cdot \log(1-h_\Theta(x^{(i)}))\right]$$

$$+ \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left[\Theta_{ji}^{(l)}\right]^2$$

↳ (for now, assume $\lambda = 0$, ignore regularization)

- focus on single example $(x^{(i)}, y^{(i)})$, case of 1 output unit.
- Consider cost: $cost^{(i)} = y^{(i)} \cdot \log h_\Theta(x^{(i)}) + (1-y^{(i)}) \cdot \log(h_\Theta(x^{(i)}))$

  $cost^{(i)} \approx (h_\Theta(x^{(i)}) - y^{(i)})^2$

  ↳ how close is $y^{(i)}$ to output $h_\Theta(x^{(i)})$?

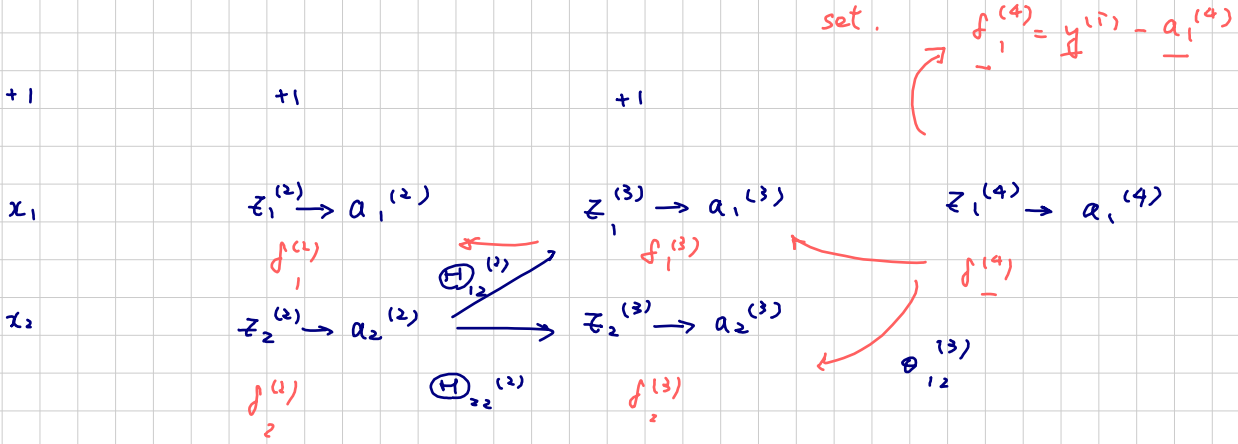
Now, $\delta_j^{(l)}$ = "error" of cost for $a_j^{(l)}$ (unit j in layer l)

Formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}^{(i)}$ for $j \geq 0$. where.

$$\text{cost}^{(i)} = y^{(i)} \log h_\Theta(x^{(i)}) + (1-y^{(i)}) \cdot \log h_\Theta x^{(i)}$$

↳ measure of how much we need to change $z_j^{(l)}$ to affect $h_\Theta(x^{(i)})$.

[partial derivates of cost func. wrt. intermediate values, so as to affect the output].



① How did we arrive at $\delta_2^{(2)}$? "Weighted sum of the activation unit in the next layer"

$$\delta_2^{(2)} = \Theta_{12}^{(2)} \delta_1^{(3)} + \Theta_{22}^{(2)} \delta_2^{(3)}$$

②. $\delta_2^{(3)} = \Theta_{12}^{(3)} \cdot \delta_1^{(4)}$.

Machine Learning Week 5 Pt. 2 : Backpropagation in Practise.

4. Implementation Note: Unrolling Parameters.

Advanced optimization

function. [jval, gradient] = costFunction (theta)
↳ $\mathbb{R}^{n+1}$.
...
↳ $\mathbb{R}^{n+1}$ (vectors)

optTheta. = fminunc. (@costFunction, initialTheta, options)

Neural Network (L=4) → not "vectors" anymore, instead "matrices"

$\Theta^{(1)}$, $\Theta^{(2)}$, $\Theta^{(3)}$ - matrices (Theta1, Theta2, Theta3)

$D^{(1)}$, $D^{(2)}$, $D^{(3)}$ - matrices (D_1, D_2, D_3)

↳ Unroll into vectors.

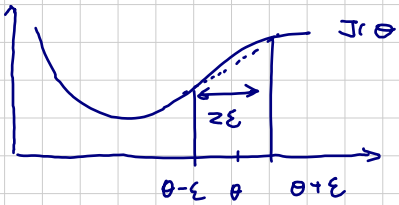
- reshape. → slice elements and fit into matrix.

Learning Algorithm.

- Have initial parameters $\Theta^{(1)}$, $\Theta^{(2)}$, $\Theta^{(3)}$ → unroll to get initialTheta.

5. Gradient Checking

- eliminates error from escaping verification. (BP is error-prone but sometimes not detectable)
- Numerical estimation of gradients.



- get secant line btwn $(\theta-\varepsilon, \theta+\varepsilon)$

$\frac{dJ(\theta)}{d\theta} \approx \frac{J(\theta+\varepsilon) - J(\theta-\varepsilon)}{2\varepsilon}$, $\varepsilon \sim 10^{-4}$.

(alt. $\frac{dJ(\theta)}{d\theta} = \frac{J(\varepsilon+\theta) - J(\theta)}{\varepsilon}$)

When. $\theta \in \mathbb{R}^n$. [unrolled version of $\Theta^{(1)}$, $\Theta^{(2)}$, $\Theta^{(3)}$]

So $\theta = (\theta_1, \theta_2, \dots \theta_n)$

$$\frac{\partial}{\partial \theta_1} J(\theta) \approx \frac{J(\theta_1+\varepsilon, \theta_2, \dots \theta_n) - J(\theta_1-\varepsilon, \theta_2, \dots \theta_n)}{2\varepsilon}$$

$$\vdots$$

$$\frac{\partial}{\partial \theta_n} J(\theta)$$

↳ approxing $\frac{\partial}{\partial \theta} J(\theta)$

change only 1. and hold other. const.

· key: Check grad $\approx$ DVec ← from backprop.

Note

very expensive.

① BP to compute DVec. [unrolled $D^{(1)}, D^{(2)}, D^{(3)}$]

② Implement numerical grad. check. to compute grad Approx.

③ Make sure they give similar values.

[D. $f^{(1)}, f^{(2)} \dots$]

(*) ④ Turn off gradient checking, use backprop. for learning.

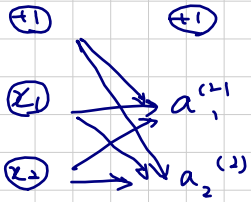
↳ Remember! Disable gradient checking else code will run slow

6. Random Initialization.

· We need initial value for Θ

- set initialTheta = zeros(n,1) → logistic ✓

- NN.



$\Theta_{ij}^{(l)} = 0 \quad \forall\, l, i, j$

After each update, weight pairs are still same (diff. value) btwn. each other.

$a_1^{(2)} = a_2^{(2)}$, also $\delta_1^{(2)} = \delta_2^{(2)}$

$$\Rightarrow \frac{\partial}{\partial \Theta_{01}^{(1)}} J(\Theta) = \frac{\partial}{\partial \Theta_{02}^{(1)}} J(\Theta) \Rightarrow \Theta_{01}^{(1)} = \Theta_{02}^{(1)}$$

⇒ Random Initialization: Symmetry breaking.

Initialize each $\Theta_{ij}^{(l)}$ to a random value in $[-\varepsilon, \varepsilon]$

(i.e. $-\varepsilon \le \Theta_{ij}^{(l)} \le \varepsilon$)

↓ not same ε as prev. sect.

7. Summary.

(1) Training a neural network.

- pick a network architecture (connectivity)

  - layers, # of act. units.

  - input unit: dimension of features $x^{(i)}$.
  - output ": # of classes.

multi class → $y = \{1, 2, 3 \dots, 10\}$.

$y = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$ or $\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$

[$y=1$] [$y=2$].

- Reasonable default.

  - 1 hidd. layer
  - ≥ 1 hidd layer, w/ same # of hidd. units.

(the more the better) ⟺ complex ?

✓

1) Randomly initialize weights
2) FP: $h_\Theta(x^{(i)})$ $\forall x^{(i)}$
3) $J(\Theta)$
4) BP: $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

- for loop over examples (get $a^{(l)}$ and $\delta^{(l)}$ $l = 2, \dots L$)

5) Gradient checking : comp. $\frac{\partial}{\partial \Theta^{(l)}_{jk}} J(\Theta)$ BP

$\updownarrow$

num. est. of $\frac{\partial}{\partial \Theta} J(\Theta)$

disable

6) < Grad. descent
Adv. opt. method. > BP to min. $J(\Theta)$ over Θ.

- $J(\Theta)$ non-convex $\Rightarrow$ can get stuck in local minima.
↳ good enough in practice.